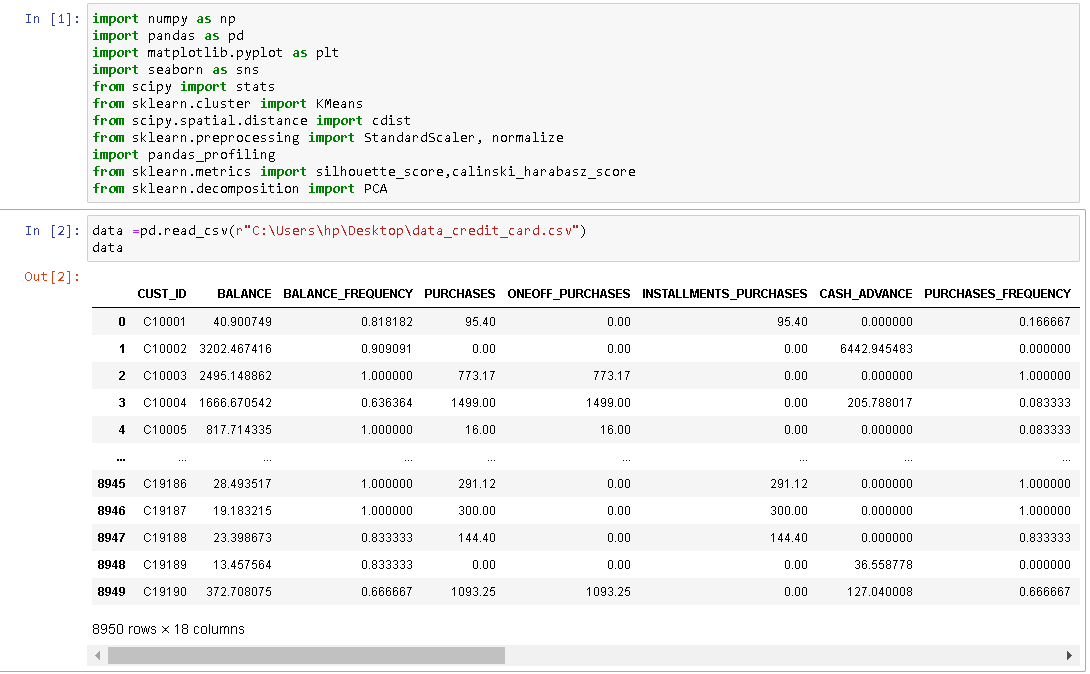
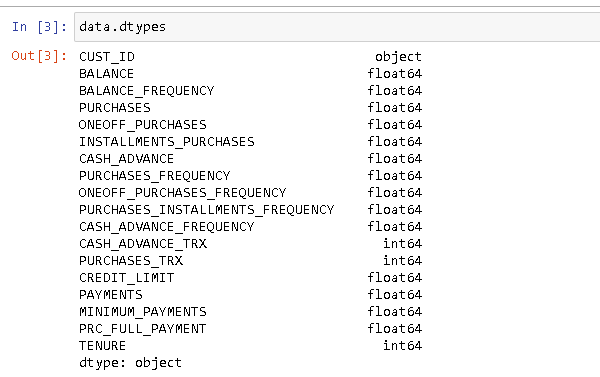
**1. Preprocessing the data ( 15 points)**

**a. Check a few observations and get familiar with the data. ( 1 points)**

****

****

Approach Used:

* Importing pandas to use functions related to datafrme operations, manipulation and analysis
* importing numpy to use built in mathematical functions
* importing pyplot from matplotlib and seaborn to plot different graphs
* importing stats from scipy to use statistical function
* importing kmeans from sklearn.cluster using which we will make kmeans model on out dataframe
* importing cdist from scipy.spatial.distance to Compute distance between each pair of the two collections of inputs
* importing functions normalize and standard scalar from sklearn.preprocessing.
* Standard scalar standardises the datases by removing the means and scaling the values to unit variance
* in addition Normalize scales the columns to unit normals
* silhouette\_score & calinski\_harabasz\_score are two metrics which can be used to compare the performance of different clusters

Inferences:

* All the data (except CUST\_ID) are in numerical form
* We will eventually drop CUST\_ID as it doesn't do anything to affect a customer falling in any cluster
* BALANCEFREQUENCY, PURCHASESFREQUENCY, ONEOFFPURCHASESFREQUENCY, PURCHASESINSTALLMENTSFREQUENCY, CASHADVANCEFREQUENCY are not real datapoints, but a metric to compare different frequencies

**b. Check the size and info of the data set. ( 2 points)**

****

Approach used:

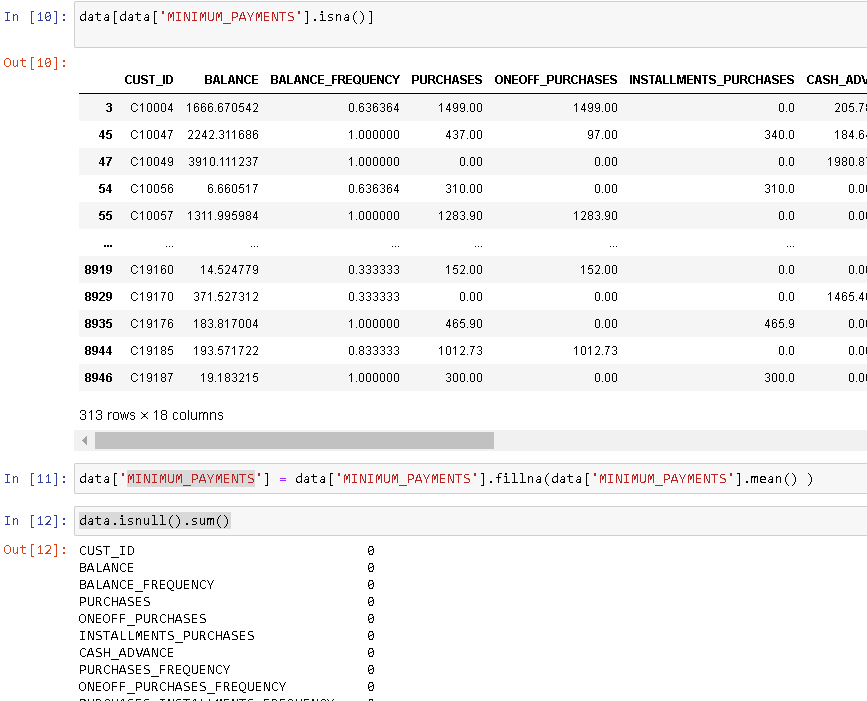
* used 'shape' to get the number of rows and columns in the dataframe
* used function info(). It gives three columns: columns names, count of non-null entries in each column & data type of each column
* 'dataframe.size' gives the total number of data points the dataframe contains

Insights

* there are 8950 rows and 18 columns (of which CUST\_ID will be dropped, so effectively we have 17 columns)
* columns CREDIT\_LIMIT and MINIMUM\_PAYMENTS contain 1 and 313 null values respectively, We need to sort it in future
* total number of individual datapoints in the dataframe are 161100

**c. Check for missing values. Impute the missing values if there are any. ( 2 points)**

****

****

Approach used

* use isnull() function that returns true for null values.
* on appending sum() with isnull(), we get total number of null (sum of Trues i.e., sum of 1's) values in each column
* first print the only row with a null value in CREDIT\_LIMIT column to see if other entries in the row are genuine or bogus
* it is done by putting a condition on the dataframe to print the row which has CREDIT\_LIMIT as null
* substitute null CREDIT\_LIMIT value with mean CREDIT\_LIMIT value using fillna() function by passing = ata['CREDIT\_LIMIT'].mean()
* fillna() function automatically finds out the rows with null values and substitutes it with the parameter (value) given in it
* print rows containing MINIMUM\_PAYMENTS as null (in the same way we did with CREDIT\_LIMIT): to get a bird's eye view of such data
* substitute the null values with mean value of MINIMUM\_PAYMENTS using the fillna() function.
* We can see using isnull().sum() that there are no null values in the entire dataframe

Insights

* CREDIT\_LIMIT contains 1 null values
* MULTIPLE\_PAYMENTS contains 313 null values

**d. Drop unnecessary columns. ( 2 points)**

****

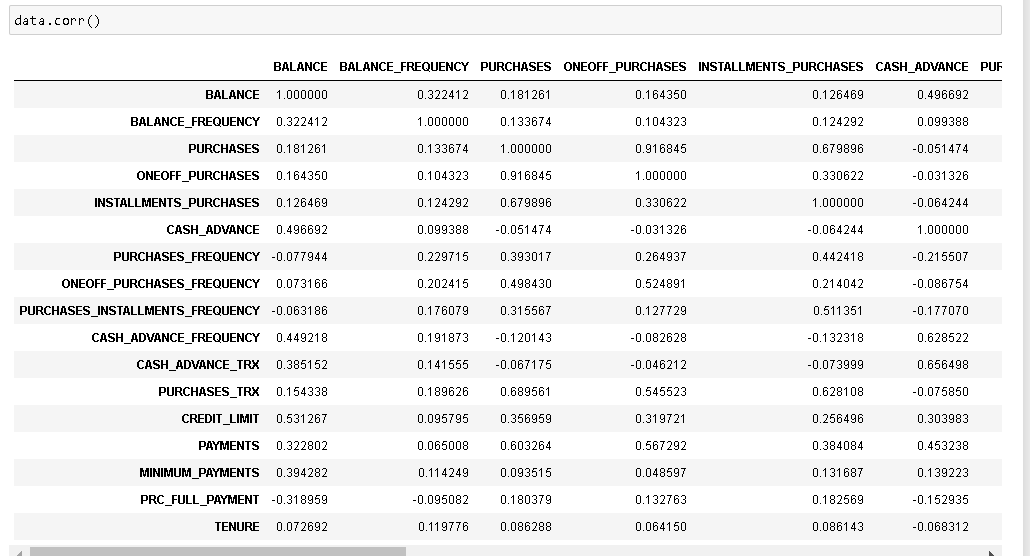
Approach Used:

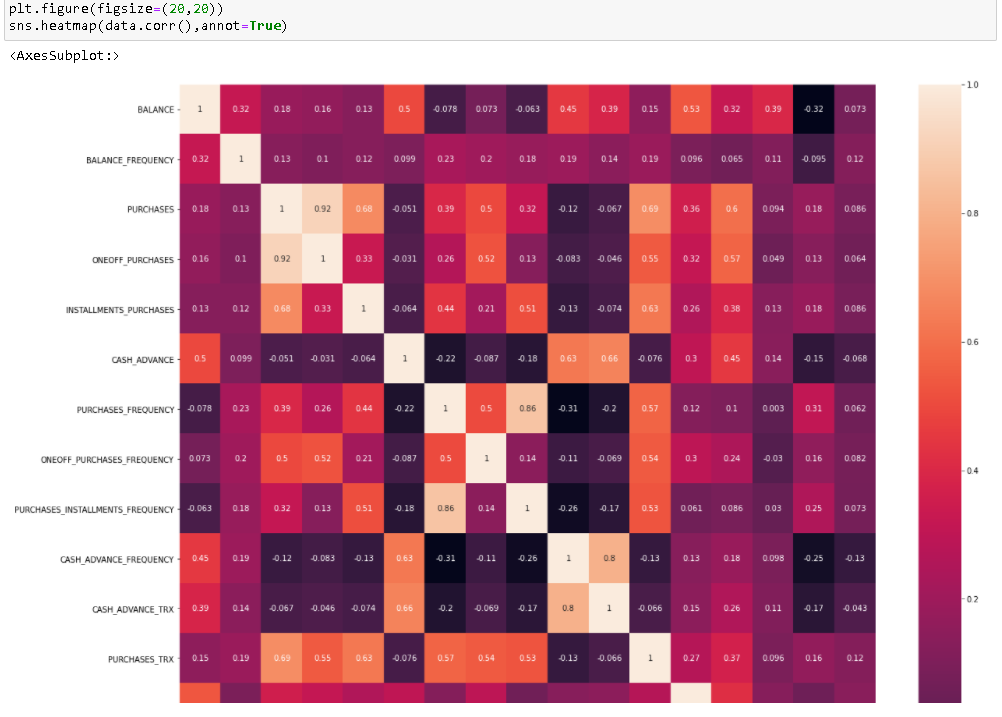
* use drop() function to drop a column. we can enter the column to be dropped and put "inplace" equal to True

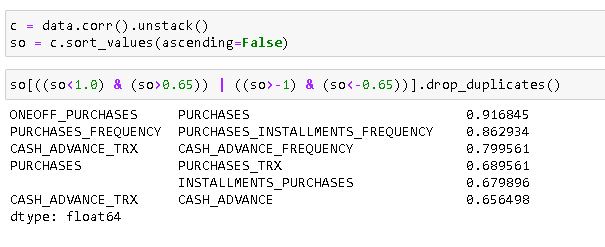
Insights:

* CUST\_ID column is unique for each customer( in this case, entry). It will not help us in model building, so we are dropping it

**e. Check correlation among features and comment your findings. (3 points)**

****

****

****

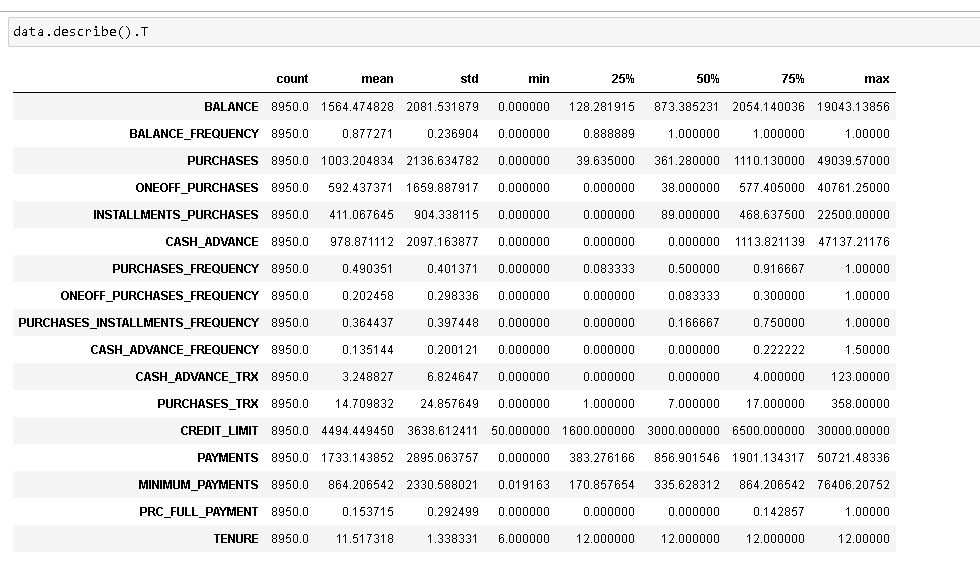
Approach used:

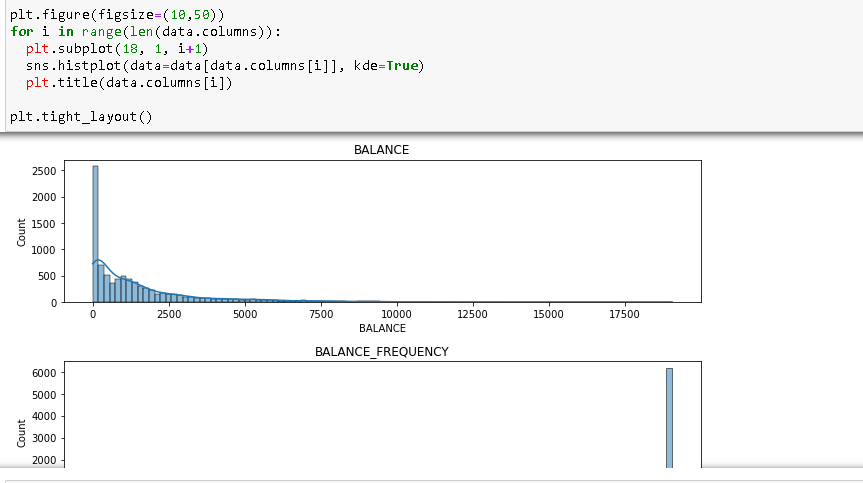
* use corr() to print correlating between every column pair of the dataframe.
* hoewever to fint independent/ highly correlated attributes we will need to look at each and every points in the correlation matrix which is tedious
* so, we use heatmap to plot correlation and find out independent (dark colored) and highly correlated(light colored) tiles
* for better view, we can use unstack() which converts the correlation matrix into individual points and columns pertaining to it in a row.
* then we can filter the correlation numbers as required and columns related to it
* in this code I have filtered columns with correlation between (1,0.65) or (-1,-0.65)
* we use drop\_duplicates so that the column pairs (which will have same correlation either way) are not repeated

Insights:

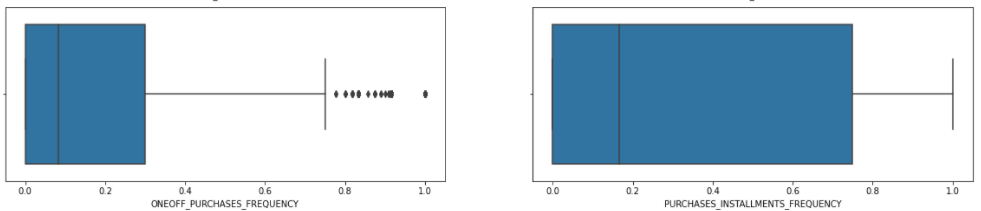
* from the filtered correlation value, we can see that (ONEOFF\_PURCHASES and PURCHASES), (PURCHASES\_INSTALLMENTS\_FREQUENCY,PURCHASES\_FREQUENCY), (CASH\_ADVANCE\_FREQUENCY, CASH\_ADVANCE\_TRX),(PURCHASES,PURCHASES\_TRX),(PURCHASES, INSTALLMENTS\_PURCHASES), (CASH\_ADVANCE, CASH\_ADVANCE\_TRX) have correlation in excess of 0.65.

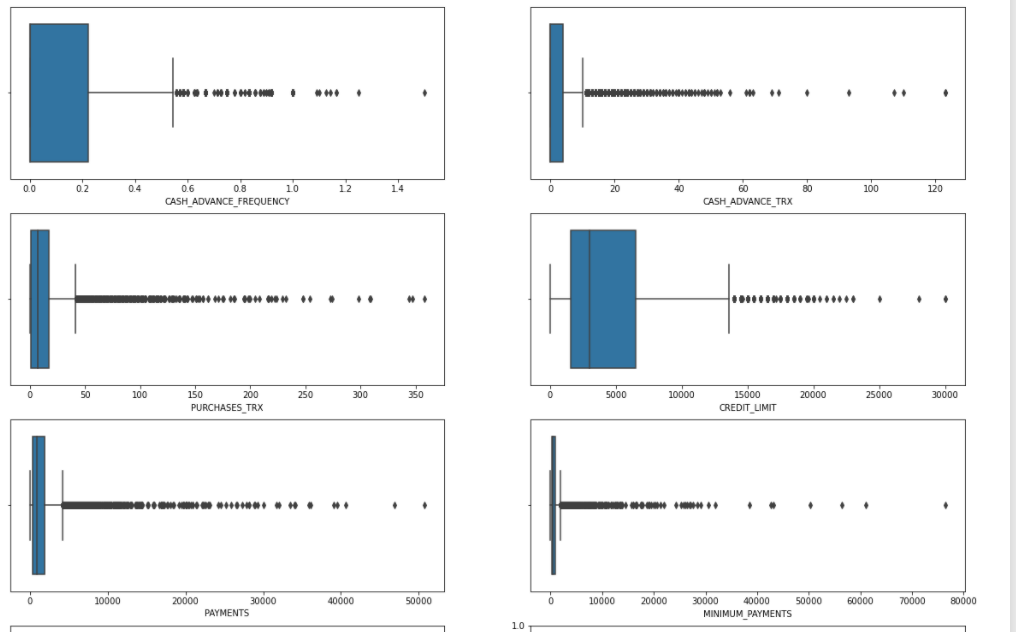
**f. Check distribution of features and comment your findings. (3 points)**

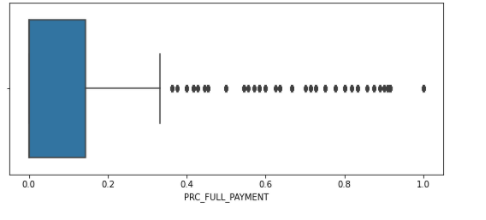
****

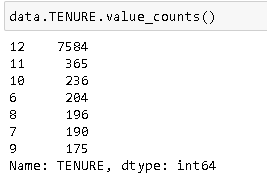
****

****

****

****

****

****

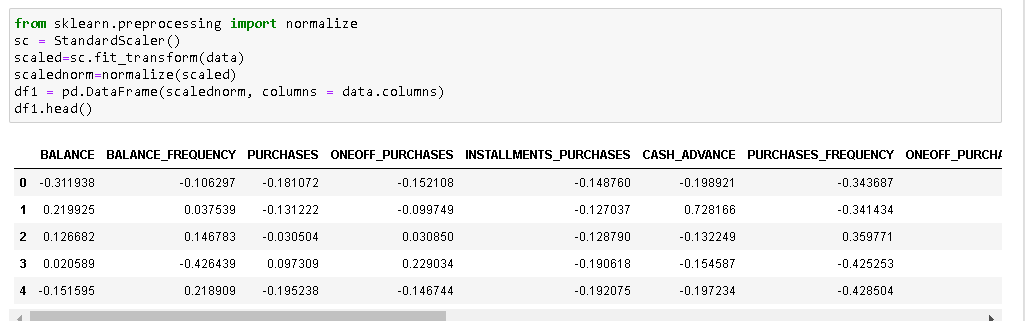
Approach used:

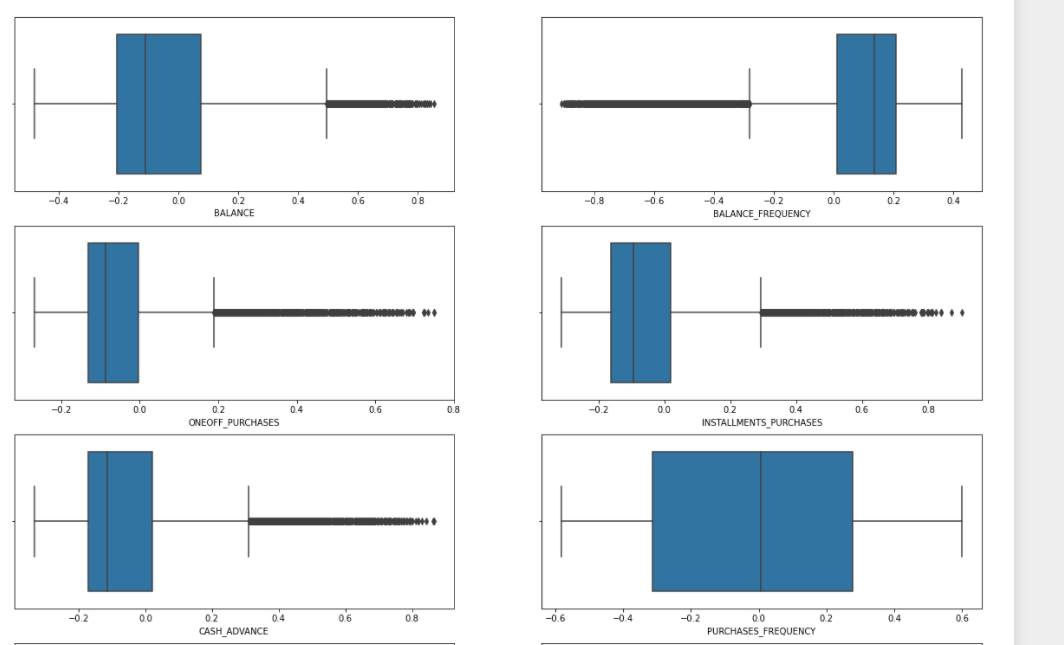
* use describe function which provied 5 points summary and standatd deviation ,mean and number of rows
* we then use profilereport() from pandas\_profiling which provides detailed report related to individual attributes along with the relation between them
* we do get normal histogram plot for individul attributes in pandas\_profiling, however, to get a magnified view, we plot each attribute individually
* do to this , we use for loop that iterates over all the columns of the dataframe and plots sns.histplot for every attribute
* along with this, we also plot boxplot for each attribute to get and idea of their distribution and presence of outliers
* TENURE column acts more as a categorical attribute, so, we use value\_counts() to view the distribution of it

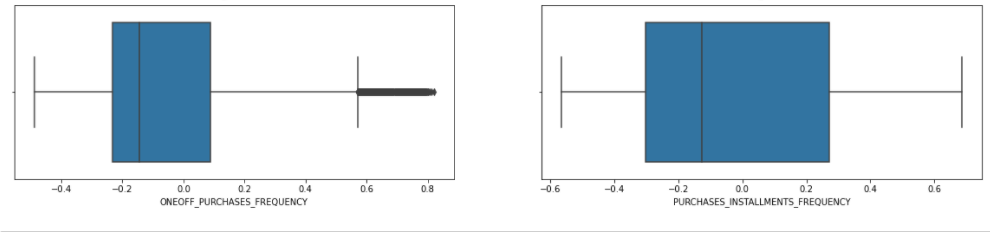
Insights:

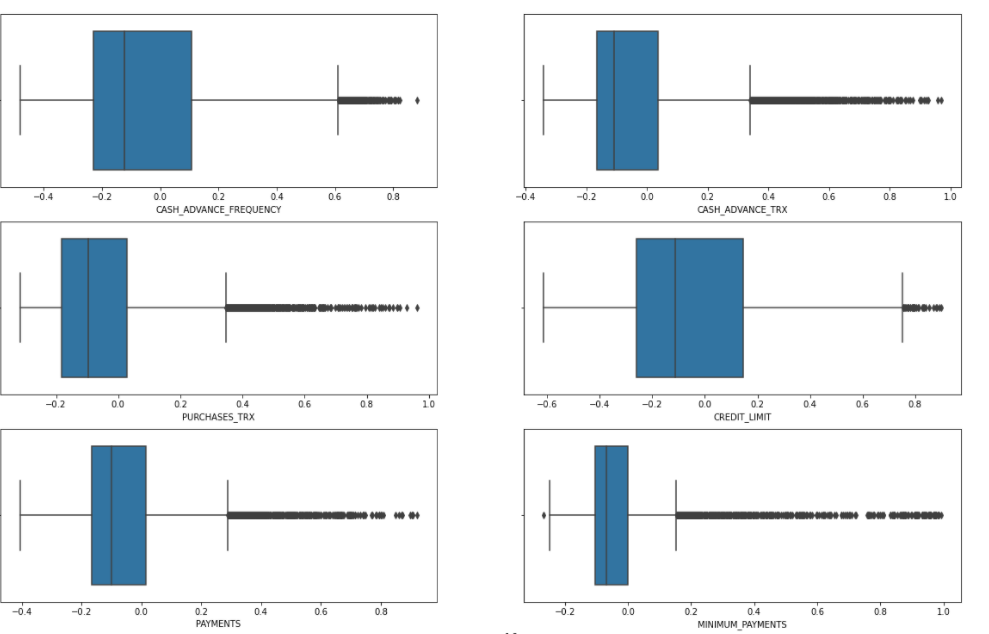
* from describe(), and the plots (pandas\_profiling, histplot and boxplot), we find out that most of the data attributes are skewed.
* columns containing absolute real value attributes like, BALANCE, PURCHASES, ONEOFF\_PURCHASES,INSTALLMENTS\_PURCHASES, CASH\_ADVANCE,CASH\_ADVANCE\_TRX, PURCHASES\_TRX,CREDIT\_LIMIT, PAYMENTS, MINIMUM\_PAYMENTS are right skewed (because most of the values are small )
* columns containing frequecy attributes like (BALANCE\_FREQUENCY,PURCHASES\_FREQUENCY, ONEOFF\_PURCHASES\_FREQUENCY, PURCHASES\_INSTALLMENTS\_FREQUENCY,CASH\_ADVANCE\_FREQUENCY) have most of the entries concentrated at the extremes (0 or 1), which implies that most of the people are fairly regular with their payments or dont pay them at all
* presence of outliers are mostly because of the skew and not due to some bogus entries (inferred because all the outliers fall on one side of the boxplot that is towards the tail)

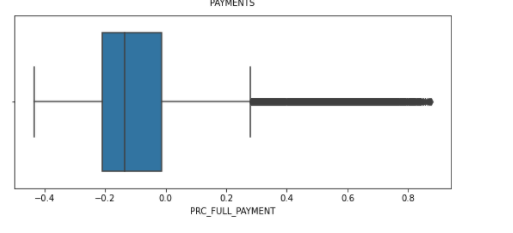
**g. Standardize the data using appropriate methods. ( 2 points)**

****

****

****

****

****

Approach Used

* use standardscaler and normalise from sklearn.preprocessing library
* Standard scalar standardises the datases by removing the means and scaling the values to unit variance
* in addition Normalize scales the columns to unit normals
* create a new dataframe df1 where we store the scaled and normalised values in accordance with the respective columns
* use describe() function to get an idea about the distribution of the new dataframe.
* plot boxplots for every attribute on df1 dataframe to view if the outliers have been treated/ reduced

Insights

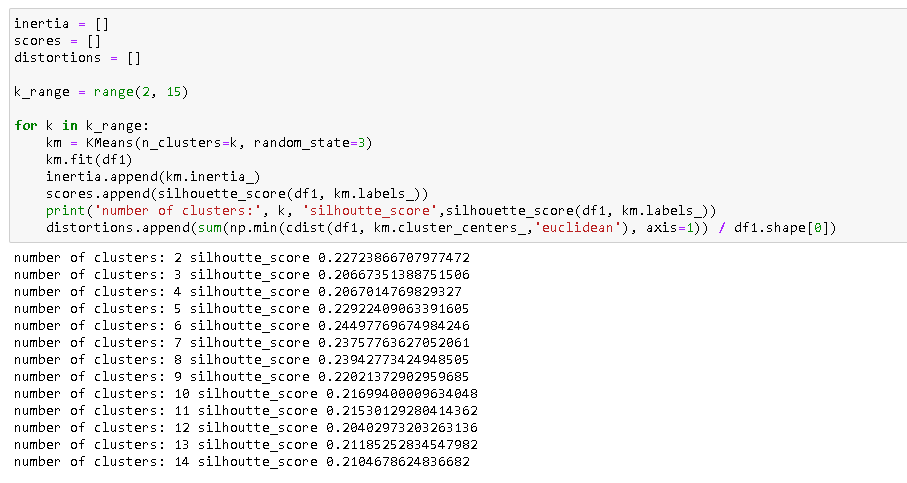
* after standardisation and normalisation all the values in every column fall between -1 and 1
* from the boxplots we can see that number of outliers has been reduced, also the extremity(distance of outlier from mean) is lesser than the non-standardised and non-normalised data

**2. Build a k-means algorithm for clustering credit card data. Kindly follow the below steps and answer the following. ( 10 points)**

**a. Build k means model on various k values and plot the inertia against various k values.**

**b. Evaluate the model using Silhouette coefficient**

**c. Plot a elbow plot to find the optimal value of k**

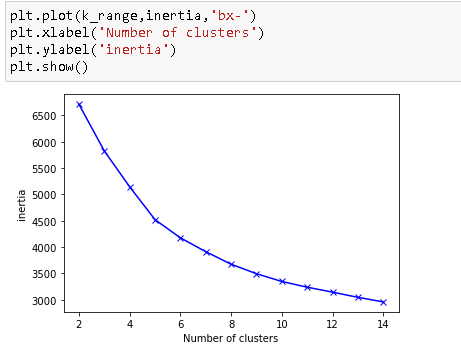
****

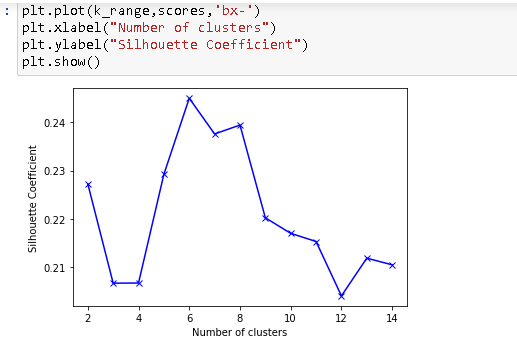
Approcah used:

* initialise three arrays inertia, scores, and distortions
* inertia contains km.inertia\_ data pertaining to each number of clusters (from 2-15). Inertia is sum of squared distance of each point from its assigned centroid. for better modelling, inertia should be minimum
* scores contains silhouette\_score. silhouette score measures the degree of separation between clusters
* silhouette score= (bi-ai)/max(bi,ai); bi=shortest mean distance between a point i to all points outside the cluster of i; ai=bi=shortest mean distance between a point i to all points inside the cluster of i
* k\_range is the range over which for loop will iterate (numbers of clusters a model will have)
* initialise the for loop and define km as KMeans model with number of cluster equalling the turn of iteration.
* fit the data df1 to model km.
* once the data df1 is fitted, all the metrics can be generated.
* append inertia\_ metrics of kmeans into inertia array, silhouette\_score metrics into scores array
* we calculate distortion (sum of squared errors of each datapoints w.r.t its cluster). it is similar to inertia and can be used interchangeably to decide the optimum number of clusters
* to calculate distortion, we use cdist funcion (that measures distance between each pair of two collected inputs).
* min() appended before cdist ensures that one cdist result is extracted that measures the distance between individual points and their respective centroids.
* append the result into distortions array

Insights:

* from the output we can infer that with 6 clusters, we have highest silhouette score.



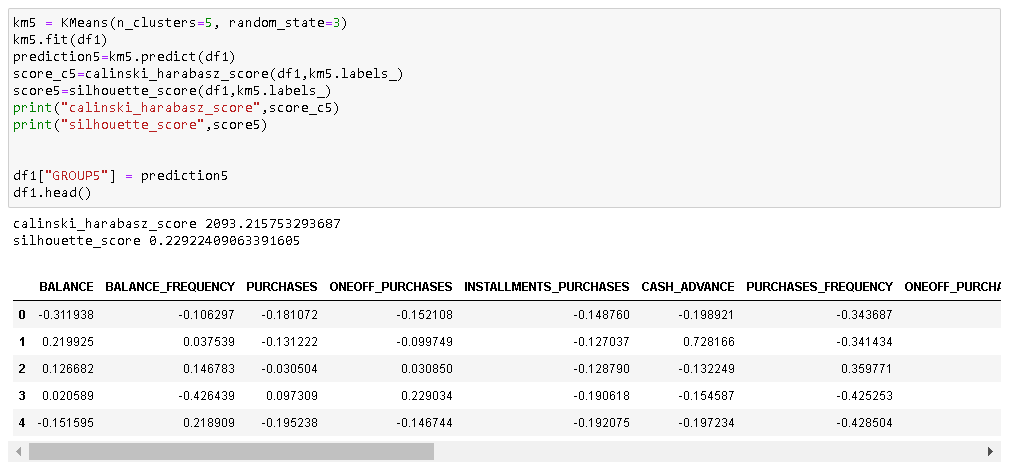


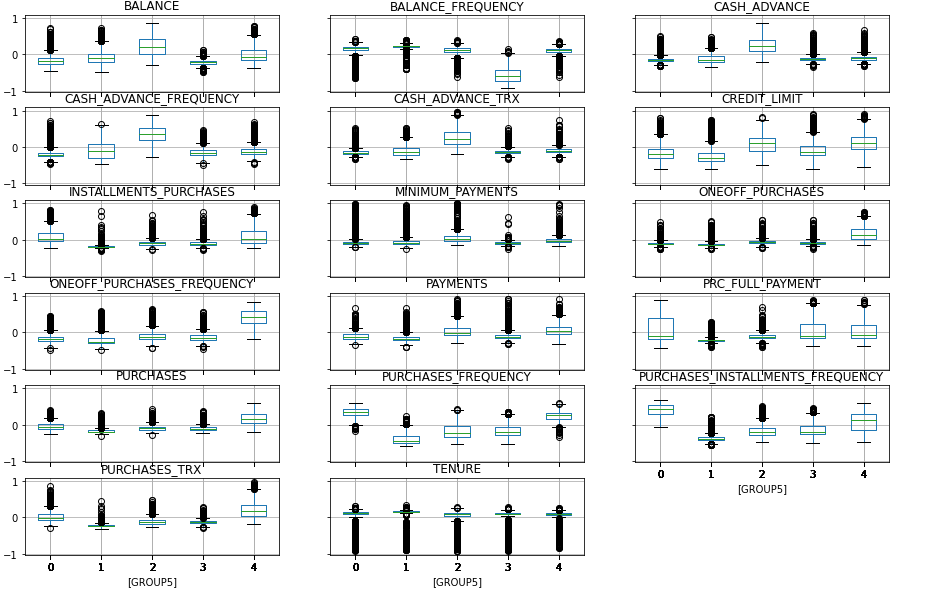


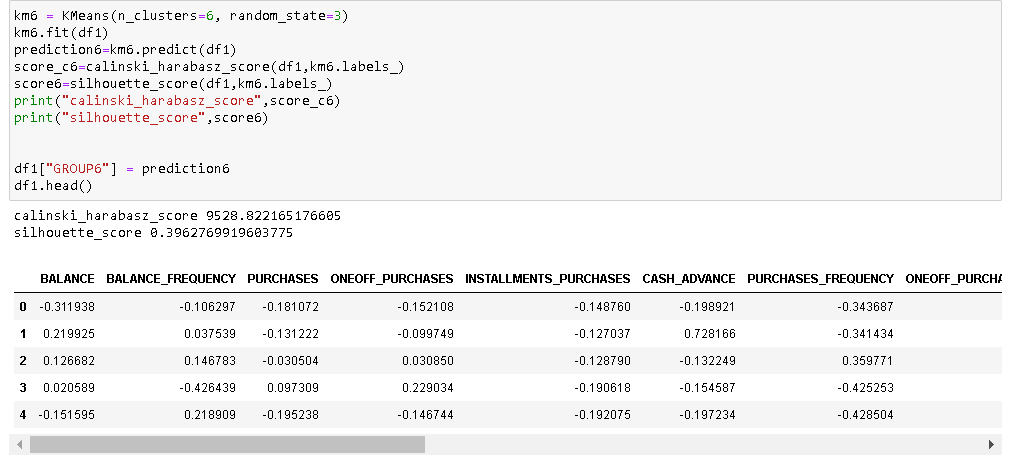
Approcah used:

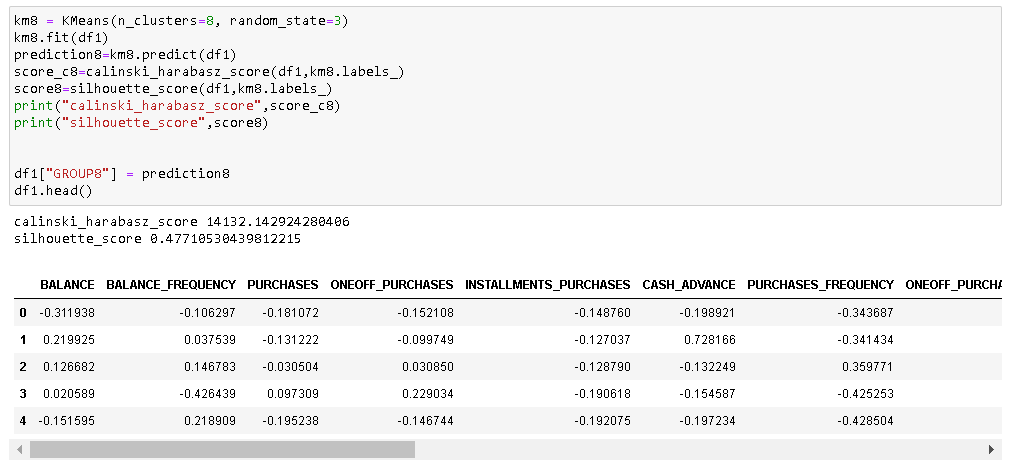
* plot inertia, silhouette scores and distortion arrays against the number of clusters by using plt.plot Insights:
* we can clearly see an elbow at 5 number of clusters. However, we can achieve better silhouette score at 6 number of clusters
* also if we use 8 clusters, we will significantly reduce the distortion, and keeping the silhouette score high

**d. Which k value gives the best result?**

****

****

****

****

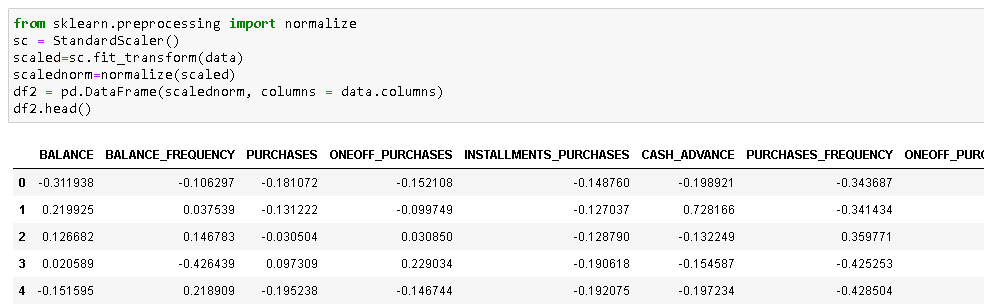
Approach used:

* try modelling with 3 different number of clusters (5,6 and 8).
* first fit the data df1 in the kmeans model and use it to predict the clusters (save the predicted values in predicted'n')
* print calinski\_harabasz\_score and silhouette\_score; append predicted values to the df1 dataframe in a column named GROUP'n'
* create a new dataframe 'clusterdata' that contains 'df1' dataframe grouped by clusters.
* clusterdata.mean gives mean of every attribute for each custer
* it can also be visualised as a boxplot

Insights:

* we can see that calinski\_harabasz\_score is in order 8>6>5 and silhouette coefficient is in order 6>8>5
* here we have concluded that having 6 clusters will be the optimum as it has fairly large silhouette coefficient and calinski\_harabasz\_score is also comparable to the highest value (to achieve 4000 points jump in calinski\_harabasz\_score we need to make two more clusters which is expensive to computation).

**3. Apply PCA to the dataset and perform all steps from Q2 on the new features generated using PCA. ( 15 points)**

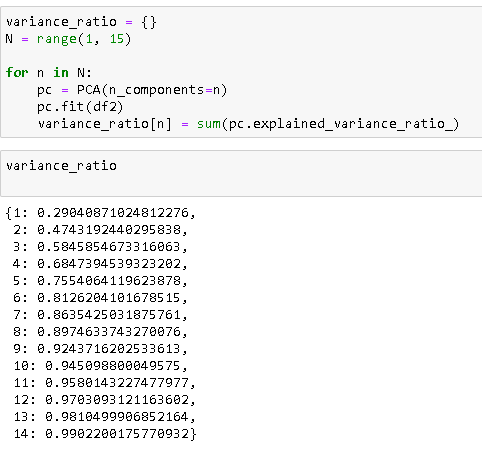
****

Approach Used

* use standardscaler and normalise from sklearn.preprocessing library
* Standard scalar standardises the datases by removing the means and scaling the values to unit variance
* in addition Normalize scales the columns to unit normals
* create a new dataframe df2 where we store the scaled and normalised values in accordance with the respective columns
* use describe() function to get an idea about the distribution of the new dataframe.

Insights

* after standardisation and normalisation all the values in every column fall between -1 and 1

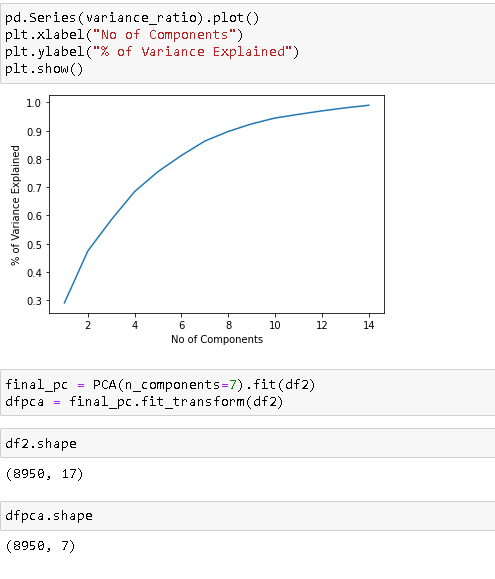
****

Approach used:

* initialize a list variance\_ratio
* initialise for loop from 1-15 and generate PCA model for each number(number of new attributes)
* fit the model with df2 data and insert the sum of variance ratio related to each (number of new attributes) into vaiance\_ratio list

Insight:

* we can see that having 7 new features on the basis of PCA, can explain as many as 87% of the variance while reducing the number of columns from 17 to 7

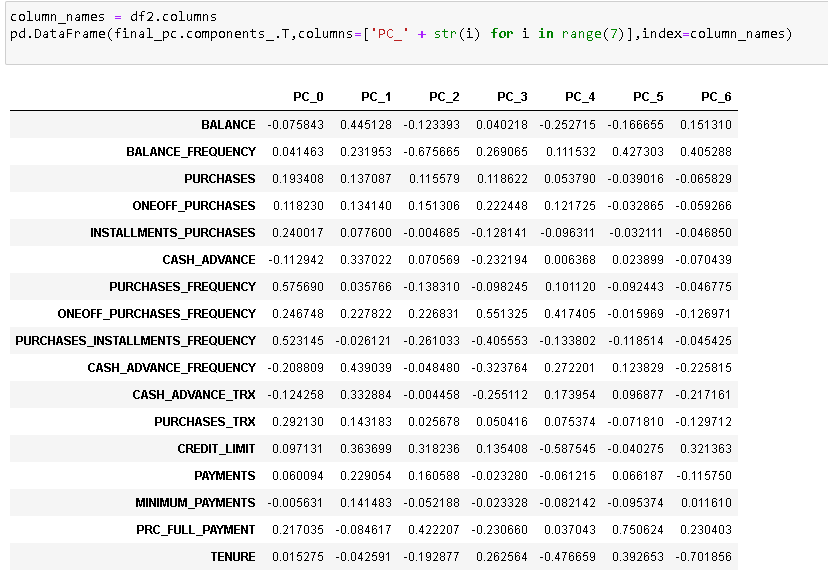
****

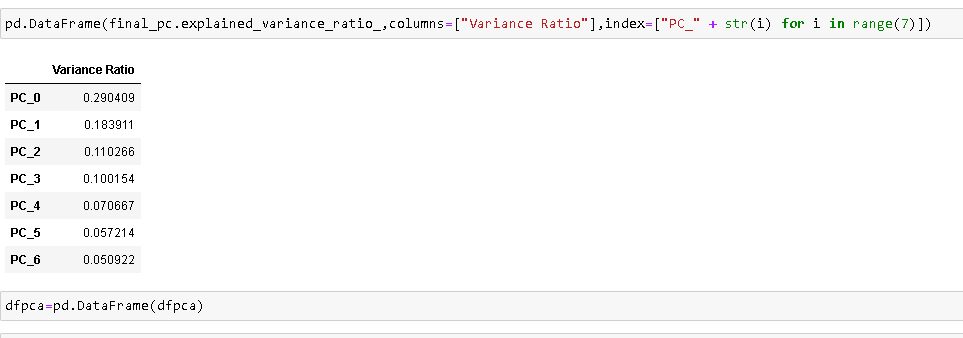
Approach used:

* use pd.series().plot() to plot the list variance ratio for better visualisation
* initialise model "final\_pc" with PCA with 7 new attributes and fit & transform the data df2
* use shape to get the changed number of attributes in the new dataframe compared to original dataframe

Insight:

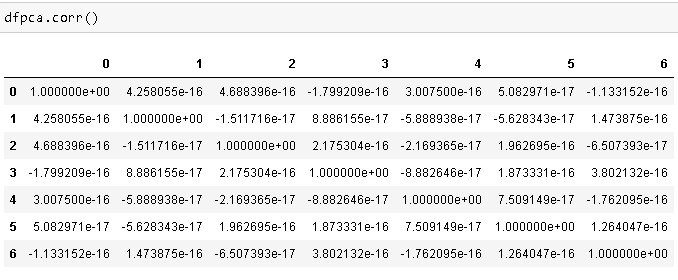
* since we didn't see a sharp vertex in the variance\_ratio plot, we select 7 as number of attributes as going from 6 to 7, we gain 5 % explaination in variance, but goint from 7 to 8, we just gain 3 %.
* changed shape of new data: 7 attributes compared to 17 sttributes in df2

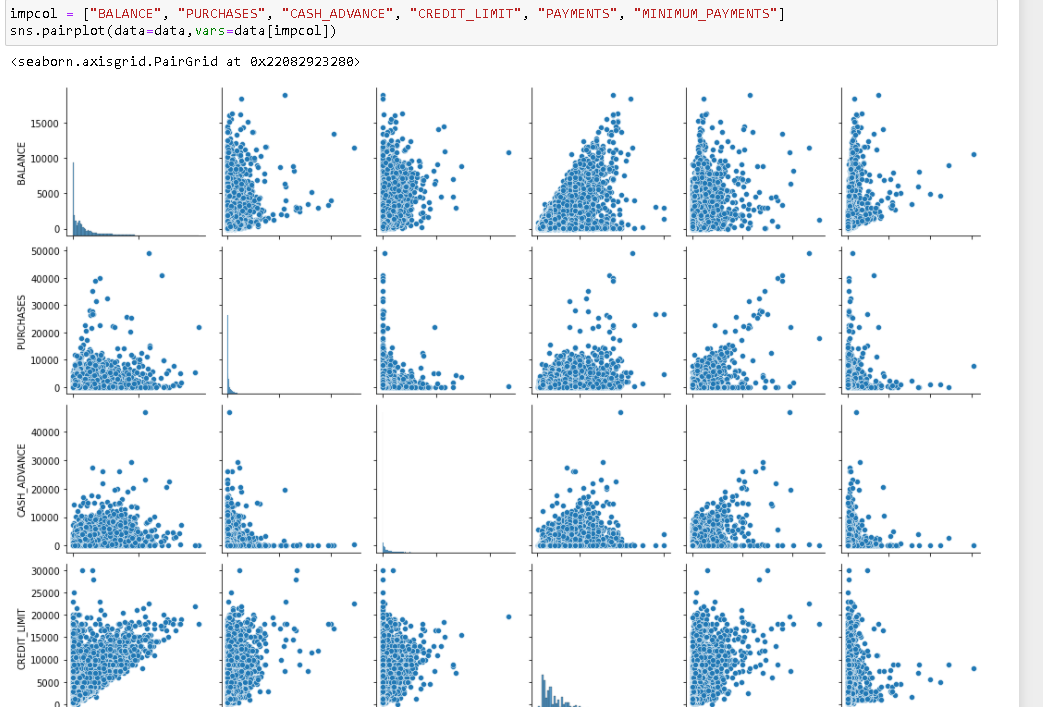
****

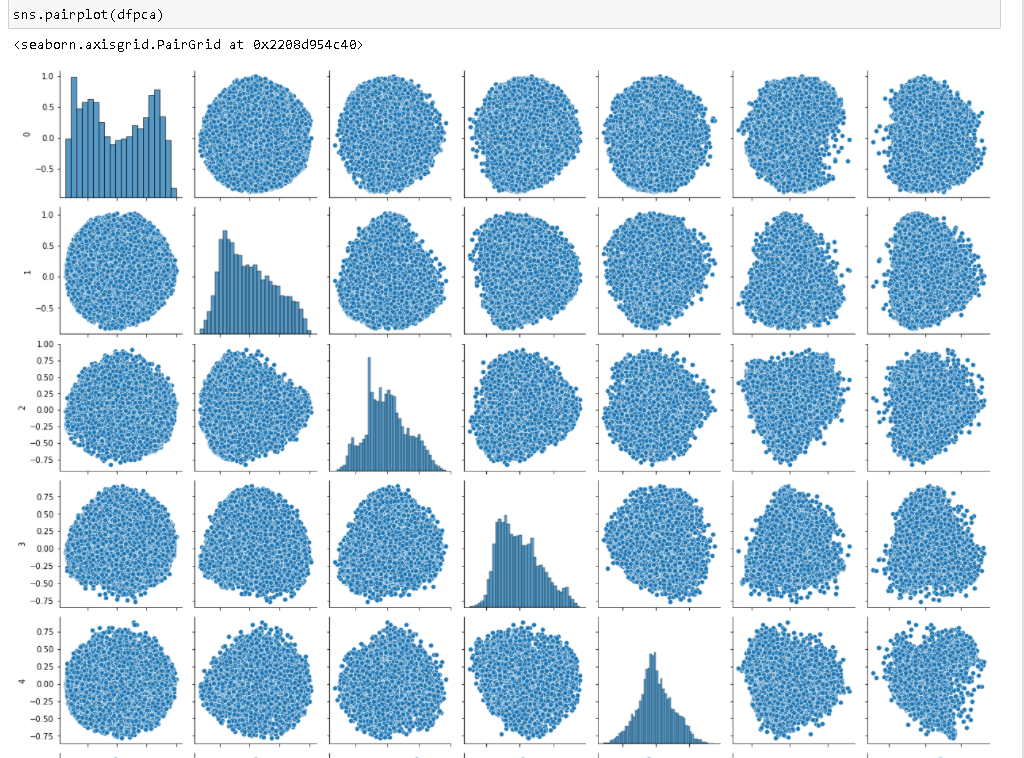
****

approach used:

* print a new dataframe that contains all the columns of df2 as index, individual attributes of PCA (dfpca) as column names and final\_pc.components\_ that contains weightage of each attribute in explaining the original attribte
* print a new dataframe that contains explained\_varaiance\_ratio (variance explained by each attribute)
* convert the dfpca into a dataframe

****

****

****

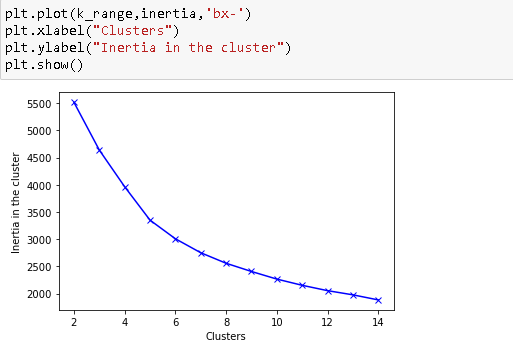
approach used:

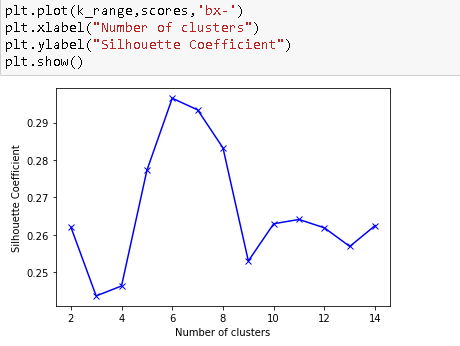
* use ProfileReport to get preliminary view of the new PCA dataframe (dfpca)
* use corr() to get correlation between every attributes of dfpca
* pair plotting original dataframe with 17 columns, takes extremely long time. SO we have used a few important columns for pairplotting
* decide the important columns, save it into an array and pass it as 'var' in sns.pairplot.
* plot pairplot of PCA (dfpca) data

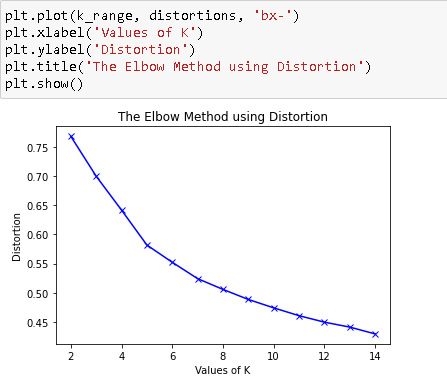
insight:

* pairplot of the original data shows substantial correlation between several attributes.
* from ProfileReport, pairplot and corr() for PCA (dfpca), we can conclude that most of the new attributes are normally distributes and have nearly 0 correlation with any other attribute

****

****

****

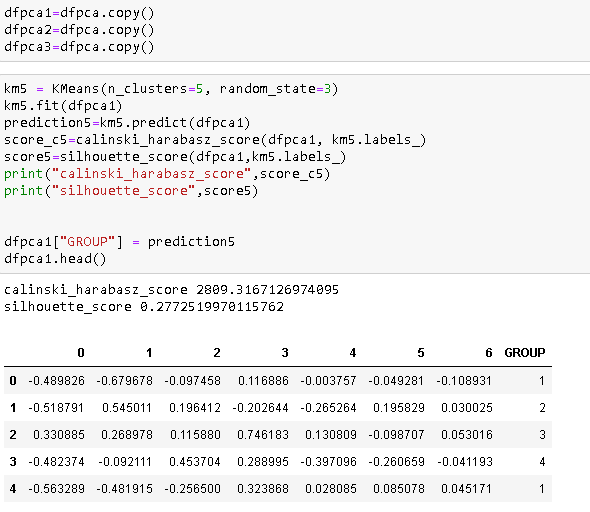
****

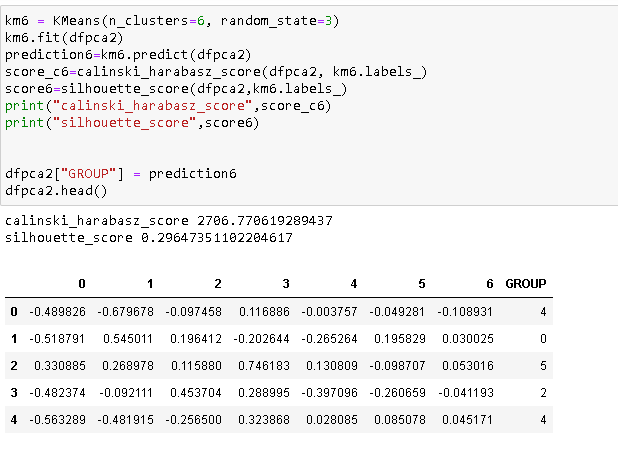
Approcah used:

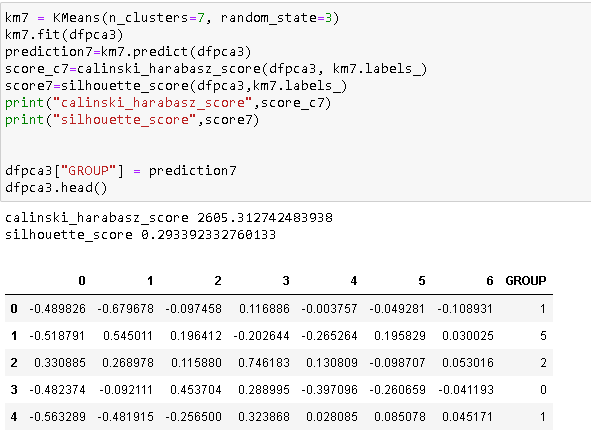
* initialise three arrays inertia, scores, and distortions
* inertia contains km.inertia\_ data pertaining to each number of clusters (from 2-15). Inertia is sum of squared distance of each point from its assigned centroid. for better modelling, inertia should be minimum
* scores contains silhouette\_score. silhouette score measures the degree of separation between clusters
* silhouette score= (bi-ai)/max(bi,ai); bi=shortest mean distance between a point i to all points outside the cluster of i; ai=bi=shortest mean distance between a point i to all points inside the cluster of i
* k\_range is the range over which for loop will iterate (numbers of clusters a model will have)
* initialise the for loop and define km as KMeans model with number of cluster equalling the turn of iteration.
* fit the data df2 to model km.
* once the data df2 is fitted, all the metrics can be generated.
* append inertia\_ metrics of kmeans into inertia array, silhouette\_score metrics into scores array
* calculate distortion (sum of squared errors of each datapoints w.r.t its cluster). it is similar to inertia and can be used interchangeably to decide the optimum number of clusters
* to calculate distortion, we use cdist funcion (that measures distance between each pair of two collected inputs).
* min() appended before cdist ensures that one cdist result is extracted that measures the distance between individual points and their respective centroids.
* append the result into distortions array
* plot inertia, silhouette scores and distortion arrays against the number of clusters by using plt.plot

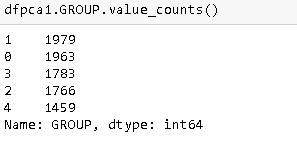
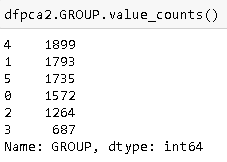
Insights:

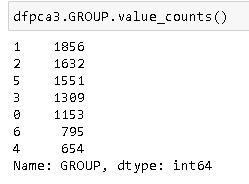
* from the output we can infer that with 6 clusters, we have highest silhouette score.
* we can clearly see an elbow at 5 number of clusters. However, we can achieve better silhouette score at 6 number of clusters

****

****

****

****

****

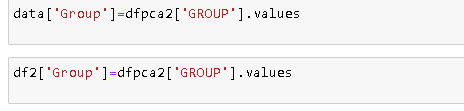
Approach used:

* create 3 copies of the PCA data dfpca
* model the data dfpca1, dfpca2, and dfpca3 with 3 different number of clusters (5,6 and 7).
* first fit the data in the kmeans model and use it to predict the clusters (save the predicted values in prediction'n')
* print calinski\_harabasz\_score and silhouette\_score; append predicted values to the dfpca'n' dataframe in a column named GROUP'n'
* print value counts; number of rows in each cluster for all the three trials

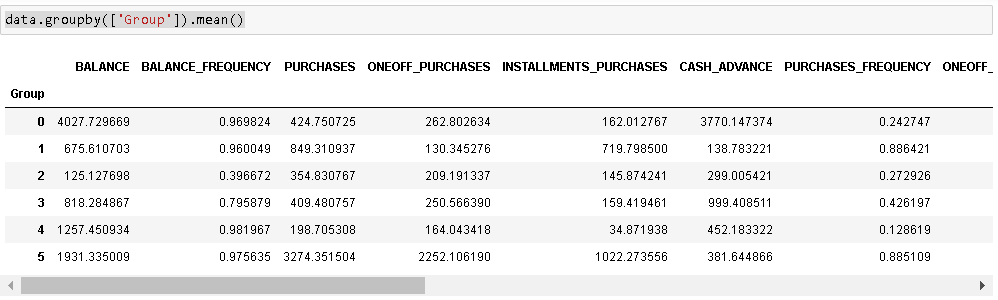
Insights:

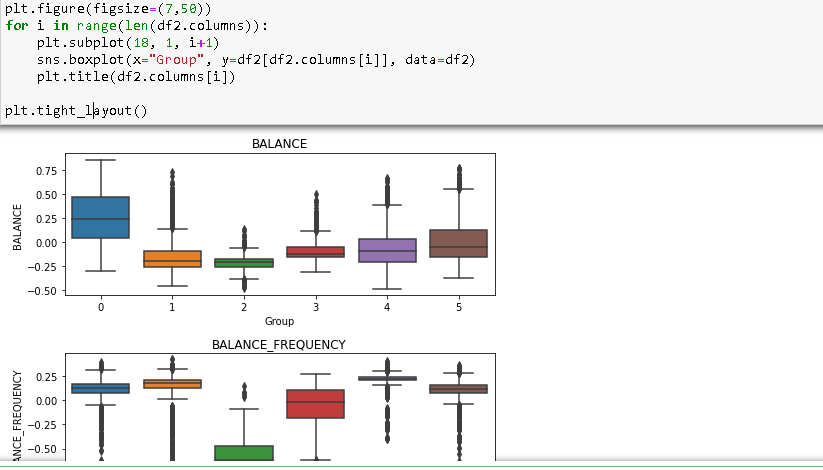
* calinski\_harabasz\_score is in the order 5>6>7
* silhouette\_score is in order 6>7>5
* We conclude that 6 is the optimal number of clusters considering the two scores

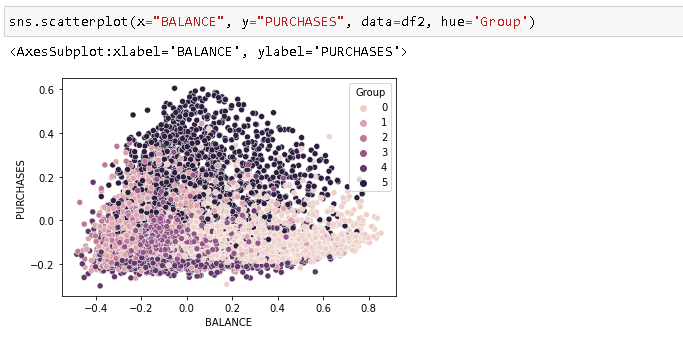
**4. Create a new column as a cluster label in the original data frame and perform cluster analysis. Check the correlation of cluster labels with various features and mention your inferences. (Hint - Does cluster 1 have a high credit limit?) ( 5 points)**

****

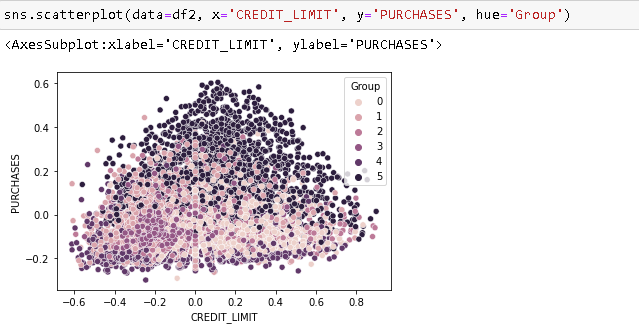
****

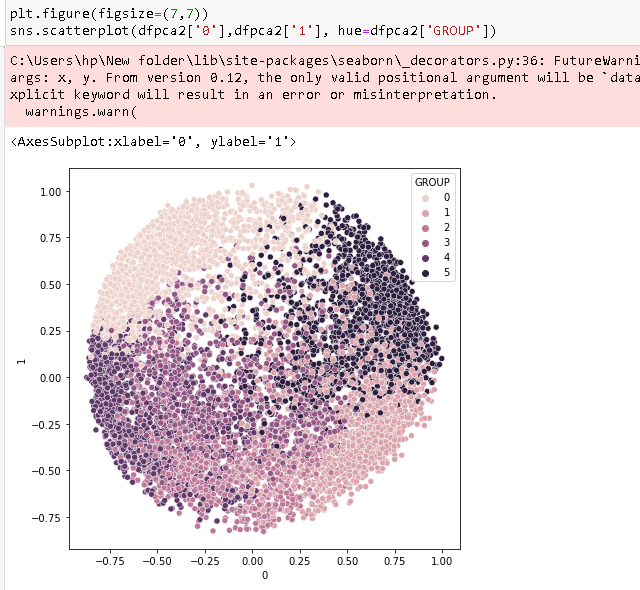
****

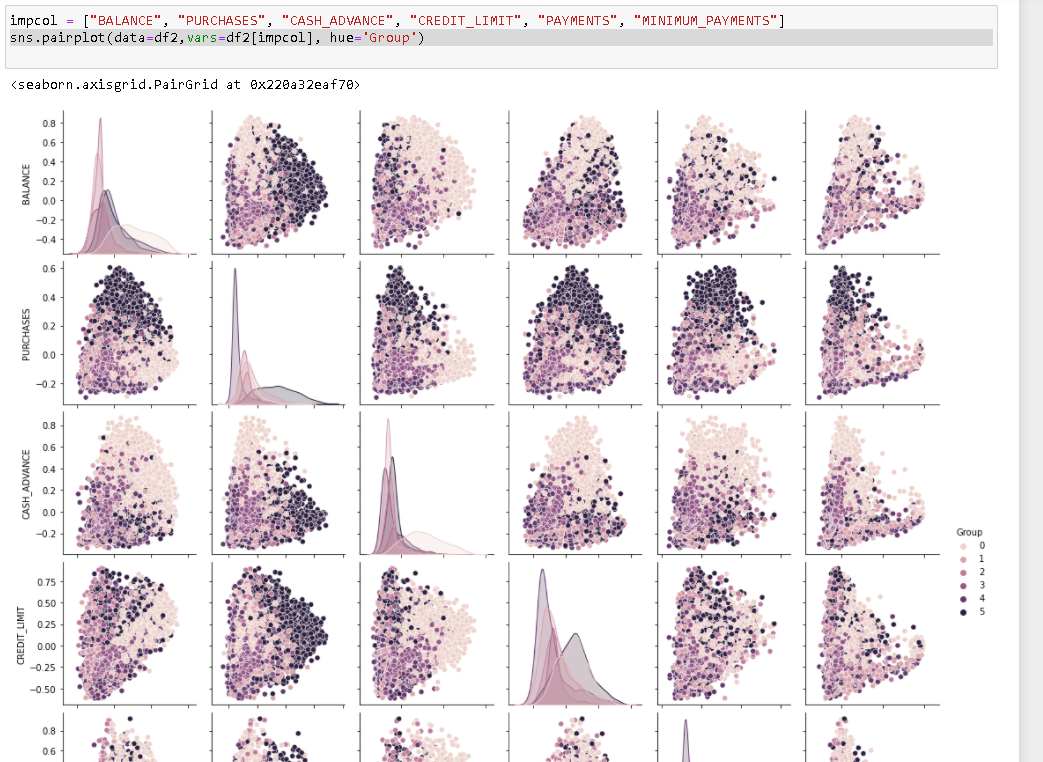
****

****

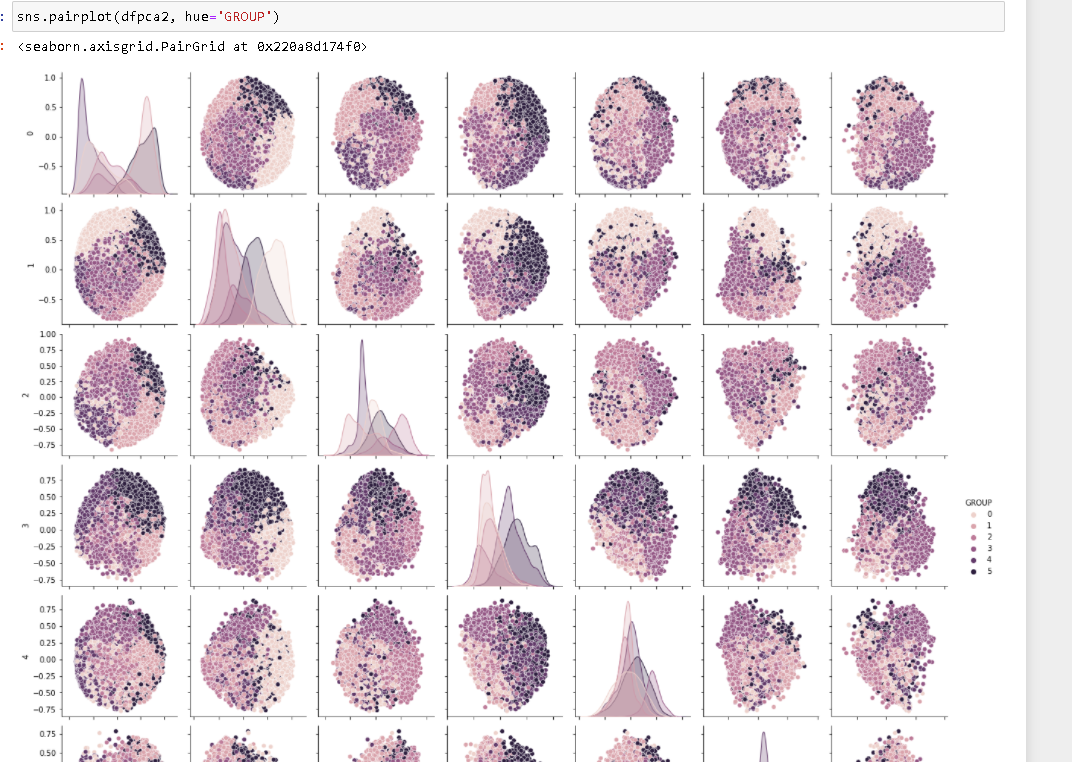
****

****

****

****

****

****

Approach used:

* add GROUP column from dfpca2 (PCA data clustered using kmeans cluster with 6 clusters) to the original dataframe 'data' and normalised+standadised dataframe df2
* use value\_counts to get number of rows in each cluster
* use groupby() function to group all the rows by 'GROUP' column and append mean() to it, which will give mean value of each attribute for each cluster
* plot boxplot for every attribute grouped by 'GROUP' on x axis. it is a graphical visualisation of groupby().mean() - plot scatter plot with several pairs of important attributes (from original data) using 'Group' as hue (to visualise clustering)
* plot scatterplot with several pairs of attributees (from dfpca) using 'Group' as hue
* pairwise plotting important columns using seaborn

Insights:

* we can see the mean value of attributes for different clusters
* the clusters can be arranged in rank for each attribute  
  BALANCE : 0>5>4>3>1>2  
  BALANCE\_FREQUENCY : 4>5>0>1>3>2  
  PURCHASES : 5>1>0>3>2>4  
  ONEOFF\_PURCHASES : 5>0>3>2>4>1  
  INSTALLMENTS\_PURCHASES: 5>1>0>3>2>4  
  CASH\_ADVANCE : 0>3>4>5>2>1  
  PURCHASES\_FREQUENCY : 1>5>3>2>0>4  
  ONEOFF\_PURCHASES\_FREQUENCY: 5>3>0>4>2>1  
  PURCHASES\_INSTALMENTS\_FREQUENCY:1>5>3>2>0>4  
  CASH\_ADVANCE\_FREQUENCY: 0>3>4>5>2>1  
  CASH\_ADVANCE\_TRX : 0>3>4>5>2>1  
  PURCHASES\_TRX : 5>2>0>3>2>4  
  CREDIT\_LIMIT : 5>0>2>1>4>3  
  PAYMENTS : 5>0>2>1>4>3  
  MINIMUM\_PAYMENTS : 0>5>1>4>3>2  
  PRC\_FULL\_PAYMENT : 1>5>2>3>0>4  
  TENURE : 4>5>1>2>0>3
* by doing scatter plot, we can visualise the distribution of different clusters. It can be spotted that points related to single cluster tend to fall in similar part of the graph

**5. Comment your findings and inferences and compare the performance. Does applying PCA give a better result in comparison to earlier? ( 5 points)**

* the first benefit of using PCA here is that the number of attributes decreased from 17 to 7 (which makes computation very efficient, particularly when there is a large data)
* too many attributes tend to have a lot of noise, and some of them are also correlated to one another.
* Using PCA we can remove a large part of noise and get fewer number of totally independent attributes, making Kmeans clustering more efficient.
* however in this case, on using PCA, we are actually getting worse silhouette score and calinski\_harabasz\_score for kmeans clustering model( with 6 clusters in both the cases). But, we also have to note that internal evaluation metrics are not very comparable across different data sets, so we cannot really compare the performances